D3 Workshop

Prince Jiang
D3 workshop reference by Mike Bostock

Requirements from TA in CSCI55200

1. Try to find solutions on everything before heading to the office.
2. Do your homework all by yourself. DO NOT use any answer from previous students.
3. Read questions and requirements carefully before you start.
4. D3 programming is a creative part. You will not get high marks if only using existing D3 layouts and algorithms.

Preface

D3 – Visualizing Data with Web Standard

It means D3 does not generate any new representation based on Web Standards

D3 – Visualizing Data mapping to Elements

Visualization requires visual encoding: mapping data to elements.
D3 - Data-Driven Documents

The name "D3" refers to the W3C Document Object Model.

Web Standards

You can refer to W3C School for more and latest Web Standards.

HTML Resources

HTML5 Spec, HTML5 for Developers, MDN, Dive Into HTML5

hello-world.html

SVG Resources

SVG Spec, MDN, D3 API Reference

hello-svg.html

<!DOCTYPE html>
<meta charset="utf-8">
<body>
Hello, world!
</body>

<!DOCTYPE html>
<meta charset="utf-8">
<svg width="960" height="500">
<text y="12">
Hello, world!
</text>
</svg>
CSS Resources

CSS Spec, Selectors Spec

JavaScript Resources

MDN, Douglas Crockford's JavaScript: The Good Parts

Let’s get started!

http://d3js.org/

hello-css.html

hello-javascript.html

hello-d3.html
A web server is required (normally) when loading external data (e.g., d3.csv).

Using a local server to skip cross-file restrictions (Windows)

`> python -m SimpleHTTPServer 8888 (Mac)`

D3 Resources

D3 API Reference, D3 Wiki, D3 Group, D3 Stack Overflow

Selections

Selectors

CSS provides a handy way to refer to specific elements.
CSS assigns style rules via selectors.

```css
pre, code {
  font-family: "Menlo", monospace;
  font-size: 48px;
}
```

Simple selectors identify elements by one facet.

```css
#code  // <any id="code">
code  // <code>
.code  // <any class="code">
[code=bar]  // <any code="bar">
code_bar  // <code><bar></code>
```

Compound selectors identify elements by two or more facets.

```css
code.bar  // <code class="bar">
code#bar  // <code id="bar">
```

The W3C standardized selector support for JavaScript.

```javascript
document.querySelectorAll("pre, code")
d3.selectAll("pre, code")
```

Selections are Arrays

D3 provides shorthand for selecting and manipulating.

Use developer tools to select, inspect, modify and experiment.
The `attr` and `style` methods set attributes and styles, respectively.

```javascript
// select all <circle> elements
var circle = d3.selectAll("circle");
// set some attributes and styles
circle.attr("cx", 20);
circle.attr("cy", 12);
circle.attr("r", 24);
circle.style("fill", "red");
```

Method chaining allows shorter (and more readable) code.

```javascript
// select all <circle> elements
// and set some attributes and styles
d3.selectAll("circle")
  .attr("cx", 20)
  .attr("cy", 12)
  .attr("r", 24)
  .style("fill", "red");
```

`selection.append` creates new elements, append, and select.

```javascript
// select the <body> element
var body = d3.select("body");
// add an <h1> element
var h1 = body.append("h1");
h1.text("Hello!");
```

With one element selected, adds one element.

```javascript
// select all <section> elements
var section = d3.selectAll("section");
// add an <h1> element to each
var h1 = section.append("h1");
h1.text("Hello!");
```

With many elements selected, adds one element to each.

Add many elements?

There is no single operation. So we have to consider data itself.
Data

Arrays

So you can represent data however you like (in JavaScript).

// A bar chart, perhaps?
var data = [1, 1, 2, 3, 5, 8];

Data can be numbers.

bar-chart.html

// A scatterplot, perhaps?
var data = [
{x: 10.0, y: 9.14},
{x: 8.0, y: 8.14},
{x: 13.0, y: 8.74},
{x: 9.0, y: 8.77},
{x: 11.0, y: 9.26}]

Data can be objects.

dot-chart.html
Data \rightarrow Elements

Use data to create multiple elements.

We want the selection “circle” to correspond to data.

```javascript
var circle = svg.selectAll("circle")
  .data(data);

The data method computes the join, defining enter and exit.

```javascript
var circle = svg.selectAll("circle")
  .data(data);

circle.enter().append("circle");
```

Appending to the enter selection creates the missing elements.

The new elements are bound to data, so we can compute attributes.

1. First, `svg.selectAll("circle")` returns a new empty selection, since the SVG container was empty. The parent node of this selection is the SVG container.
2. This selection is then joined to an array of data, resulting in three new selections that represent the three possible states: enter, update, and exit. Since the selection was empty, the update and exit selections are empty, while the enter selection contains a placeholder for each new datum.
3. The update selection is returned by `selection.data`, while the enter and exit selections hang off the update selection; `selection.enter` thus returns the enter selection.
4. The missing elements are added to the SVG container by calling `selection.append` on the enter selection. This appends a new circle for each data point in the SVG container.
Enter, Update & Exit

When initializing, you might ignore update and exit.

```
var circle = svg.selectAll("circle")
  .data(data);
  .enter().append("circle");
  .attr("cx", x)
  .attr("cy", y)
  .attr("r", 2.5);
```

When updating, you might ignore enter and exit.

```
var circle = svg.selectAll("circle")
  .data(data);
  .attr("cx", x)
  .attr("cy", y)
  .attr("r", 2.5);
```

Enter

New data, for which there were no existing elements.

Update

New data that was joined successfully to an existing element.

Exit

Existing elements, for which there were no new data.
Key Function

You can control the join; by default, the join is by index.

// A scatterplot, perhaps?
var data = [
  {name: "Alice", x: 10.0, y: 9.14},
  {name: "Bob", x: 8.6, y: 8.14},
  {name: "Carol", x: 13.0, y: 8.74},
  {name: "Dave", x: 9.0, y: 8.77},
  {name: "Edith", x: 11.0, y: 9.26}
];

If needed, data should have a unique key for joining.

function key(d) { return d.name; }

var circle = svg.selectAll("circle")
  .data(data, key)
  .attr("cx", x)
  .attr("cy", y)
  .attr("r", 2.5);

The key function returns a unique string for each datum.

Loading Data

D3 provides several convenience routines using XMLHttpRequest.

CSV

Comma-Separated Values: d3.csv

stocks.csv

symbol, date, price
S&P 500, Jan 2000, 1394.46
S&P 500, Feb 2000, 1366.42
S&P 500, Mar 2000, 1498.58
S&P 500, Apr 2000, 1452.43
S&P 500, May 2000, 1420.6
S&P 500, Jun 2000, 1454.6
S&P 500, Jul 2000, 1430.83
CSV is untyped, so coercion from strings is required.

```
var format = d3.time.format("%b %Y");
d3.csv("stocks.csv", function(stocks) {
  stocks.forEach(function(d) {
    d.price = +d.price;
    d.date = format.parse(d.date);
  });
});
```

`area-chart.html`

JSON is typed, but you must still parse dates.

```
var format = d3.time.format("%b %Y");
d3.json("stocks.json", function(stocks) {
  stocks.forEach(function(d) {
    d.date = format.parse(d.date);
  });
});
```

`Stocks.json`

Data is Asynchronous

Code that depends on data must be invoked via callback.
Data is Messy

Data is rarely in the exact format needed for visualization.

array.

\{filter, map, sort, \ldots\}

JavaScript has a number of useful built-in array methods.

d3.

\{nest, keys, values, \ldots\}

D3 also has a variety of data-transform methods; explore the API.

Scales & Axes

Scales

Attributes

Attributes (and styles) control position and appearance.
function x(d) {
  return d * 42 + "px";
}

Domain -> Range

Scales are convenient but optional; you can roll your own.

Quantitative Scales

Map a continuous (numeric) domain to a continuous range.

var x = d3.scale.linear()
  .domain([12, 24])
  .range([0, 720]);
x(16); // 240

A linear scale simply translates and scales.

var x = d3.scale.sqrt()
  .domain([12, 24])
  .range([0, 720]);
x(16); // 268.9056992603583

A sqrt (or pow) scale applies an exponential transform.

var x = d3.scale.log()
  .domain([12, 24])
  .range([0, 720]);
x(16); // 298.82699948076737

A log scale applies a logarithmic transform.
Domains & Ranges

Typically, domains are derived from data while ranges are constant.

Use `d3.min` and `d3.max` to compute the domain.

```
var x = d3.scale.linear()
  .domain([0, d3.max(numbers)])
  .range([0, 720]);
```

Use `d3.extent` to compute the min and max simultaneously.

```
var x = d3.scale.log()
  .domain([d3.extent(numbers)])
  .range([0, 720]);
```

Use an accessor function to derive a numeric value for objects.

```
function value(d) { return d.value; }
var x = d3.scale.log()
  .domain(d3.extent(objects, value))
  .range([0, 720]);
```

Interpolators

Quantitative scales support multiple interpolators.

Colors are detected automatically for RGB interpolation.

```
var x = d3.scale.linear()
  .domain([12, 24])
  .range(["steelblue", "brown"]);
x(16); // #666586
```
String interpolation matches embedded numbers; quite flexible.

```javascript
var x = d3.scale.linear()
  .domain([12, 24])
  .range(["0px", "720px"]);
x(16); // 240px
```

Interpolators can be set explicitly, if desired.

```javascript
var x = d3.scale.linear()
  .domain([12, 24])
  .range(["steelblue", "brown"]);
  .interpolate(d3.interpolateHsl);

x(16); // #3cb05f
```

**Interpolating Objects**

**Diverging Scales**

*Demo*

*Sometimes*, you want a compound ("polylinear") scale.

```javascript
var x = d3.scale.linear()
  .domain([-10, 0, 100])
  .range(["red", "white", "green"]);
x(-5); // #ff8080 x(50); // #80c080
```

The domain and range can have more than two values!

**Ordinal Scales**

Map a discrete domain to a discrete range.
An ordinal scale is essentially an explicit mapping.

```javascript
var x = d3.scale.ordinal()
    .domain(["A", "B", "C", "D"])  
    .range([0, 10, 20, 30]);
x("B"); // 10
```

Ordinal scales are often used to assign categorical colors.

```javascript
var x = d3.scale.category20()
    .domain(["A", "B", "C", "D"]);
x("B"); // #aec7e8
```

Unknown values are implicitly added to the domain.

```javascript
var x = d3.scale.category20()
    .domain(["A", "B", "C", "D"]);
x("E"); // #2ca02c
x.domain(); // A, B, C, D, E
```

A handful of color scales are built-in; see also [ColorBrewer](#).  

```javascript
var x = d3.scale.ordinal()
    .domain(["A", "B", "C", "D"])  
    .rangePoints([0, 720]);
x("B"); // 240
```

Ordinal ranges can be derived from continuous ranges.

```javascript
var x = d3.scale.ordinal()
    .domain(["A", "B", "C", "D"])  
    .rangeRoundBands([0, 720], .2);
x("B"); // 206, bar position
x.rangeBand(); // 137, bar width
```

Ordinal ranges are particularly useful for bar charts.
Axes

D3 provides convenient labeling for scales.

```javascript
var yAxis = d3.svg.axis()
  .scale(y)
  .orient("left");
```

Create an axis for a given scale, and configure as desired.

```javascript
svg.append("g")
  .attr("class", "y axis")
  .call(yAxis);
```

Render the axis by calling a `<g>` selection.

```javascript
.axis path, .axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}
```

Customize axis appearance via CSS.

Ticks

Quantitative scales can be queried for "human-readable" values.
var x = d3.scale.linear()
  .domain([12, 24])
  .range([0, 720]);

x.ticks(5); // [12, 14, 16, 18, 20, 22, 24]

The requested count is only a hint (for better or worse).

---

**Tick Formats**

See d3.format and d3.time.format.

---

**Marks**

---

**SVG Coordinates**

Absolute positioning; the origin (0,0) is the top-left corner!

---

```
var svg = d3.select("body").append("svg")
  .attr("width", outerWidth)
  .attr("height", outerHeight);

var g = svg.append("g")
  .attr("transform", "translate("
    + margin.left + ",
    + margin.top + ")")

Use margins for decorative elements, such as axes.
```

---

Use transforms to define a new origin.
Hello

SVG Paths

Vertical alignment of text is slightly awkward.
Path Generators

Configurable functions for generating paths from data.

```javascript
<svg xmlns="http://www.w3.org/2000/svg">
  <path d="M152.64962091501462,320.560078085569133.891395506318,325
  .4363117213358314.96890954453846,33.83.3797634921996131.93482695327
  86,331.15839364812198.56581109628815,335.53933887857004151.144507994
  85135,335.706620232782.1830103121918,323.747339700815666.93723455
  785742,322.85068144615262,373139115540866.331.200066843387162.24833
  4309137434,335.36772270840515.84344098888326,335.4574956954036153
  97662112141221,331.3607812563317515.38052318118711,329.9479943151851
  4541.803074022372,326.624165808908568.3761003200155,326.836822370
  2955168.8237742209793313.8011259143528733.34."/>
</svg>

```javascript
var x = d3.scale.linear()
y = d3.scale.linear();

var line = d3.svg.line()
  .x(function(d) { return x(d.x); })
  .y(function(d) { return y(d.y); });
```

Compose scales with data accessors to define position.

```javascript
svg.append("path")
  .datum(objects)
  .attr("class", "line")
  .attr("d", line);
```

Pass data to the line generator directly, or via selection.attr.

```javascript
d3.svg.area

Define a path in terms of x, y0 and y1.
```
Line Interpolators

Line and area generators support multiple interpolation modes.

For non-stacked area charts, $y_0$ is constant.

```javascript
var x = d3.scale.linear(),
    y = d3.scale.linear();

var area = d3.svg.area()
    .x(function(d) { return x(d.x); })
    .y0(height)
    .y1(function(d) { return y(d.y); });
```

For streamgraphs, use d3.layout.stack to compute the baseline.
Layouts

Data

Layouts are reusable algorithms that generate data, not display.

Radial Area & Lines

Similar to d3.svg.arc, except in polar coordinates.

d3.svg.arc

A path generator for pie and donut charts, among other uses.

Override the accessors to set constant properties.

\[\text{var arc = d3.svg.arc()}
\text{.innerRadius(0)}
\text{.outerRadius(360);}\]

Use d3.layout.pie to compute start and end angles from data.

// construct a default pie layout
\[\text{var pie = d3.layout.pie();}\]

// derive data to feed to d3.svg.arc
\[\text{var myArcs = pie(numbers);}\]
Layouts are Varied

Each layout is different. Most are stateless, but not all.

Hierarchical Layouts

There are lots of ways to visualize hierarchical data!
The hierarchical layouts use a shared representation of data.

```javascript
var parent = {"children": []},
    child = {"value": ...};
```

Layouts are configurable functions.

```javascript
var treemap = d3.layout.treemap()
     .padding(4)
     .size([width, height]);
```

The layout populates position properties on the nodes.

```javascript
function x(d) { return d.x; }
function y(d) { return d.y; }
function dx(d) { return d.dx; }
function dy(d) { return d.dy; }
```

By accessing those properties, you can visualize the layout.

```javascript
svg.selectAll("cell")
     .data(treemap.nodes(root))
     .enter().append("rect")
     .attr("class", "cell")
     .attr("x", x)
     .attr("y", y)
     .attr("width", dx)
     .attr("height", dy);
```

Demo